Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/EP04/013797

International filing date: 03 December 2004 (03.12.2004)

Document type: Certified copy of priority document

Document details: Country/Office: DE
Number: 103 57 257.0
Filing date: 08 December 2003 (08.12.2003)

Date of receipt at the International Bureau: 03 February 2005 (03.02.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

BUNDESREPUBLIK DEUTSCHLAND

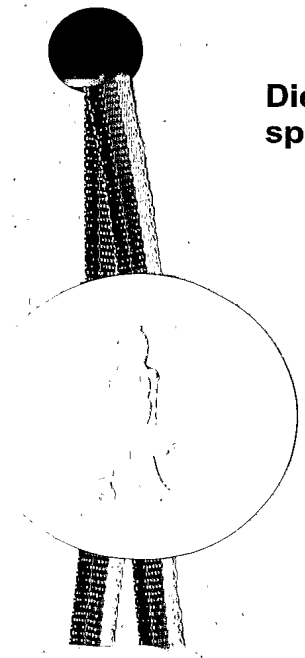
PCT/EP2004/013797
07.01.2005

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen:	103 57 257.0
Anmeldetag:	08. Dezember 2003
Anmelder/Inhaber:	Giesecke & Devrient GmbH, 81677 München/DE
Bezeichnung:	Java Smart Card Chip mit für globale Variablen reserviertem Speicherbereich
IPC:	G 06 K 19/07

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 30. Dezember 2004
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag



A 9161
03/00
EDV-L

Java Smart Card Chip mit für globale Variablen reserviertem Speicherbereich

Die Erfindung betrifft einen Smart Card Chip mit einer virtuellen Java-Card-Maschine und einem für globale Variablen reservierten Speicherbereich. Weiter betrifft die Erfindung ein Modul und eine Smart Card mit einem derartigen Chip sowie ein Verfahren zum Implementieren eines Java Programmcodes, insbesondere Java Pakets, in einen solchen Smart Card Chip.

Smart Cards, d.h. Chipkarten mit Mikroprozessor, werden bereits heute und künftig voraussichtlich noch verstärkt bei einer Vielzahl von Anwendungen eingesetzt, beispielsweise bei Mobilgeräten wie z.B. Mobiltelefonen als SIM-Karten bzw. USIM-Karten, als Bankkarten oder elektronische Geldbörsen im elektronischen Zahlungsverkehr, als Gesundheitskarten für Versicherte von Krankenversicherungen (Patientenkarte) und Ärzte (Ärztekarte), als Bürgerkarten, oder als Multiapplikationskarten, in denen mehrere der genannten oder anderer Funktionalitäten implementiert sind.

Ein Smart Card Chip hat eine Mehrzahl von Speicherbereichen, nämlich den nichtflüchtigen, nur einmal beschreibbaren ROM, den nichtflüchtigen, wiederbeschreibbaren EEPROM und den flüchtigen, wiederbeschreibbaren RAM. Alternativ können Teile des ROM und/oder des EEPROM durch Flash-Speicher ersetzt sein.

Bei der Herstellung des Smart Card Chips wird zunächst durch den Chiphersteller ein ROM-Maske genannter Programmcode-Anteil, der vor allem das Betriebssystem enthält, in den ROM implementiert. Anschließend wird, in der Regel durch den Smart Card Hersteller, der den Chip mit der implementierten ROM-Maske vom Chiphersteller bezieht, die Komplettierung der Smart Card durchgeführt, bei der Ergänzungen zum Betriebssystem und

Anwendungen des Smart Card Herstellers in den EEPROM implementiert werden. Nach erfolgter Komplettierung ist der Smart Card Chip fertig zur Herausgabe an den Kunden.

Um plattformunabhängige und gut gegeneinander abgesicherte Anwendungen zu erstellen, eignen sich objektorientierte Programmiersprachen, insbesondere Java™ der Firma Sun Microsystems Inc., sehr gut. Die Laufzeitumgebungen objektorientierter Programmiersprachen wie Java™ sind jedoch in der Regel zu umfangreich, um sie ohne Weiteres in einen Smart Card Chip implementieren zu können.

Die Java Card™ Technologie der Firma Sun Microsystems Inc., die z.B. in der aktuellen - und laufend aktualisierten - Version in dem Dokument "Java Card™ 2.2 Runtime Environment (JCRE) Specification" (derzeit verfügbar unter http://java.sun.com/products/javacard) dargelegt ist, stellt eine abgewandelte Java Technologie für Laufzeitumgebungen mit beschränkten Systemressourcen dar, die sich auch für Smart Cards eignet.

Die in der Java Card (genauer im Chip) vorgesehene Laufzeitumgebung gemäß der JCRE Spezifikation umfasst zumindest die virtuelle Java-Card-Maschine (Java Card Virtual Machine) (JCVM) und die Java Card Application Programming Interface (API) Klassen, und ggf. noch weitere Komponenten. Im Zusammenhang mit der Erfindung wird unter der Virtuellen Maschine der Java Card oder virtuellen Java-Card-Maschine (im engeren Sinn) der On-Card-Anteil der virtuellen Java-Card-Maschine (im weiteren Sinn) verstanden. Der Begriff der virtuellen Maschine (VM) wird also im engeren Sinn aufgefasst und gleichgesetzt mit dem auf der Java Card vorgesehenen

On-Card-Anteil der VM. Neben dem On-Card-Anteil kann zusätzlich ein Off-Card-Anteil der VM im weiteren Sinn vorgesehen sein.

Ein erstellter Programm-Quellcode (Source Code) eines vorbestimmten Programms, z.B. eines auf eine Java Card™ zu ladenden Programms, wird bei der Java™ Card Technologie zunächst, wie bei der Java™ Technologie, mittels eines Compilers kompiliert, so dass eine Klassendatei erzeugt wird, die das Format eines Zwischencodes, nämlich des Java Bytecode, hat, der durch die virtuelle Java-Maschine interpretierbar ist. Anschließend wird bei der Java™ Card Technologie, im Unterschied zur Java™ Technologie, der Bytecode der Klassendatei zusätzlich mittels eines Konverters in einen konvertierten Bytecode in einer cap-Datei (cap = Card Application Protocol) konvertiert. Die Compilierung und Konvertierung wird außerhalb der Java Card, „Off-Card", durchgeführt. Die cap-Datei, und somit letzten Endes das vorbestimmte Programm, wird auf die Java Card geladen, und der Bytecode der cap-Datei wird gelinkt. Beim Linken werden u.a. Zugriffsmöglichkeiten des mit der cap-Datei auf die Java Card geladenen Programms auf andere in der Java Card vorhandene Programmcode-Elemente eingerichtetd.h. es werden Verbindungen zwischen den einzelnen Packages hergestellt. Beim Linken ist zu unterscheiden zwischen dem Offcard-Linker und dem Oncard-Linker. Der Oncard-Linker ist in der Java Card (im Chip) implementiert und ist ohne weitere Komponenten funktionsfähig. Der Offcard-Linker benötigt zum Linken eine Export-Datei, die Informationen für das Linken enthält, und die nicht mit in die Java Card (den Chip) geladen wird. Die virtuelle Java-Card-Maschine interpretiert schließlich den konvertierten und gelinkten Bytecode der cap-Dateien und führt ihn aus.

In der Regel ist Java-Programmcode in Form von Paketen (packages) abgelegt. Jedes Paket enthält einen Teil des Programmcodes, z.B. Daten oder ein oder mehrere einzelne Programme. Die einzelnen Programme können entweder Systemfunktionen des Betriebssystems (in kompilierter Form: Systemklassen) oder Anwendungen sein. Die Strukturierung des Programmcodes in Pakete ist auch in der konvertierten cap-Datei beibehalten.

In der cap-Datei enthält jedes Paket neben dem eigentlichen Programmcode zusätzlich eine Import-Komponente und eine Export-Komponente.

In der Import-Komponente ist festgelegt, welche Zugriffsmöglichkeiten das Paket auf andere Pakete benötigt, z.B. welche Methoden aus anderen Paketen das Paket benutzen möchte. Die benötigten Zugriffsmöglichkeiten werden beispielsweise durch eine Referenz „import<anderes Paket>" in der Import-Komponente angegeben. Dabei wird eine solche „import<...>"-Referenz, die der Form nach eine Namensreferenz (Referenz, die auf einen Namen gerichtet ist, z.B. auf „anderes Paket") ist, bei der Erzeugung der cap-Datei i.d.R. in ein sogenanntes Token umgewandelt, das der Form nach eine Nummernreferenz (Referenz, die auf eine Nummer gerichtet ist) ist.

In der Export-Komponente ist festgelegt, welche Zugriffsmöglichkeiten das Paket anderen Paketen bietet, d.h. z.B. welche Methoden das Paket anderen Paketen zur Benutzung zur Verfügung stellt. Die zur Verfügung gestellten Zugriffsmöglichkeiten werden beispielsweise durch die Angabe einer Adresse in der Export-Komponente erzielt, wobei durch die Adresse der Speicherort des zur Verfügung gestellten Programmcodes angegeben ist.

Dass ein neu in eine Java Card geladenes Paket tatsächlich auf ein vorbestimmtes anderes Paket zugreifen und dessen Programmcode nutzen kann, wird durch das Linken eingerichtet. Dabei, beim Linken, lädt das neu geladene Paket Linkinformation (z.B. eine Adresse) aus der Export-Komponente des anderen Pakets in die eigene Import-Komponente. Beispielsweise wird beim Linken ein Token, d.h. eine aus einer „import"-Referenz erzeugte Nummernreferenz, in der Import-Komponente des neuen Pakets durch eine Adresse aus der Export-Komponente des anderen Pakets ersetzt. Hierdurch wird der Token (die Referenz) mit dem Benutzungswunsch durch eine tatsächliche Adress-Verknüpfung zwischen den beiden Paketen ersetzt. Die Verknüpfung und damit die tatsächliche Benutzungsmöglichkeit kann nur eingerichtet werden, wenn dem neu geladenen Paket die Export-Komponente des anderen Pakets, dessen Programmcode das neue Paket nutzen will, zur Verfügung steht.

Die Export-Komponenten aller Pakete, die in einer Java Card implementiert sind, oder eine vorbestimmte Teilmenge aller dieser Export-Komponenten, sind in der Export-Datei zusammengefasst. Wird vor der Komplettierung der Java Card ein zusätzliches (neues) Paket in die Java Card nachgeladen, wird es unter Verwendung des Offcard-Linkers und der Export-Datei gelinkt. Nach der Komplettierung der Java Card kann zum Linken nur der Oncard-Linker verwendet werden.

Anwendungen sind in der Regel in Form von Applets in der Java Card implementiert, wobei die Applets wiederum zu Anwendungs-Paketen gruppiert sind. Bei den Anwendungs-Paketen gibt es sogenannte preloaded packages (preissuance packages), die vor oder bei der Komplettierung in den Smart

Card Chip implementiert werden, und die in der Regel durch den Smart Card Hersteller implementiert werden. Weiter gibt es postloaded packages (postissuance packages), die nach erfolgter Komplettierung in den Smart Card Chip geladen werden, in der Regel durch den Abnehmer oder Kunden, beispielsweise ein Kreditinstitut oder eine Behörde.

Die Inhalte (z.B. Applets bzw. Systemfunktionen) unterschiedlicher Pakete sind durch Firewalls gegeneinander abgesichert, wohingegen die Inhalte (Applets etc.) innerhalb ein und desselben Pakets nicht mittels Firewalls gegeneinander abgesichert sind.

Java stellt eine Reihen von unterschiedlichen Variablentypen zur Verfügung. Z.B. gibt es eine Reihe von nicht globalen, auf ihren Kontext beschränkten und dynamisch erzeugten Variablen, z.B. Instanzvariablen oder gleichbedeutend Objektvariablen (instance variables), lokale Variablen (local variables) und Methodenparameter (method parameters). Instanzvariablen, lokale Variablen, Methodenparameter haben gemeinsam, dass sie dynamisch während der Laufzeit eines Programms mit dem Objekt (= Instanz einer Klasse), dem Ablauf (z.B. Programmschleife), der Methode etc. erzeugt werden, zu dem sie gehören und nur so lange existieren wie das Objekt, der Ablauf bzw. die Methode existiert, d.h. sie sind transient. Zudem sind sie nicht global, d.h. sie sind nur innerhalb des Kontexts (d.h. innerhalb des Objekt, des Ablaufs bzw. der Methode) verwendbar, in bzw. mit dem sie erzeugt worden sind. Über den Kontext hinaus verhindert die Firewall, dass die Variablen verwendet werden, d.h. der Zugriff auf die Variablen von außerhalb des Kontext ist durch die Firewall verhindert.

Der für Instanzvariablen reservierte Speicherbereich liegt auf dem heap-Speicher, im nichtflüchtigen EEPROM (oder alternativ im Flash-Speicher). Der für lokale Variablen reservierte Speicherbereich liegt auf dem Stack-Speicher, im RAM, und hat nur eine kurze Lebensdauer, nämlich die Dauer der Methoden-Ausführung.

Der Nachteil der nicht globalen Variablen ist, dass sie außerhalb ihres Kontext nicht verwendbar sind.

Neben den bereits genannten Variablen gibt es als die einzigen globalen (s.u.) Variablen bei Java die Klassenvariablen, die statisch erzeugte Variablen sind (class variables, static variables).

Klassenvariablen (gleichbedeutend verwendeter Begriff: `static` Variablen) sind globale Variablen, d.h. sie sind nicht an einzelne Instanzen einer Klasse (= einzelne Objekte) gebunden. Klassenvariablen werden statisch deklariert, wobei der für die Klassenvariablen reservierte Speicherbereich im EEPROM liegt, also in einem nichtflüchtigen Speicherbereich der Java Card.

Schreibzugriffe auf den EEPROM sind langsam und kosten daher viel Zeit. Aus diesem Grund ist der Zugriff auf Klassenvariablen langsam, weshalb die Performance eines Programms, das Klassenvariablen verwendet, gering ist. Zum Anderen erlaubt ein EEPROM lediglich eine Höchstzahl von typischerweise 100 000 Schreibzyklen, weshalb es wünschenswert ist, Schreibzugriffe auf den EEPROM wo möglich zu vermeiden. Die Verwendung von Klassenvariablen, insbesondere in Fällen, wenn nur eine kurze Lebensdauer der Variablen erforderlich ist, verlangsamt somit die Performance des Pro-

grammablaufs und belastet durch die dabei erfolgenden Schreibprozesse den EEPROM.

Andererseits besteht der Bedarf, Variablen global, also unabhängig vom Kontext eines speziellen Objekts, Pakets, einer speziellen Methode etc. zu verwenden, beispielsweise für den Austausch von Variablen zwischen unterschiedlichen Kontexten (Objekten, Paketen, Methoden) etc.. Derzeit ist diese Möglichkeit nur durch Klassenvariablen gegeben, die die weiter oben genannten Nachteile haben, wie starke Belastung des EEPROM und Langsamkeit.

Aufgabe der Erfindung ist es daher, einen Smart Card Chip mit einer virtuellen Java-Card-Maschine und einem für globale Variablen reservierten Speicherbereich zur Verfügung zu stellen, der einen einfachen, schnellen und für den Chip schonenden Zugriff auf die globalen Variablen ermöglicht.

Die Aufgabe wird gelöst durch ein Variablen-System nach Anspruch 1. Vorteilhafte Ausgestaltungen der Erfindung sind in den abhängigen Ansprüchen angeführt.

Der Smart Card Chip gemäß Anspruch 1 hat einen für globale Variablen reservierten Variablen-Speicherbereich, der im flüchtigen Arbeitsspeicher reserviert ist, beispielsweise im RAM. Zugriffe auf den flüchtigen Arbeitsspeicher sind schnell und zudem wird dadurch, dass der flüchtige Arbeitsspeicher verwendet wird, der nichtflüchtige Anwendungsspeicher, insbesondere EEPROM oder ein Flash-Speicher, geschont. Hierdurch ermöglicht der Chip

gemäß Anspruch 1 einfachen, schnellen und für den Chip schonenden Zugriff auf die globalen Variablen.

Daher ist gemäß Anspruch 1 ein Smart Card Chip mit einer virtuellen Java-Card-Maschine und einem für globale Variablen reservierten Speicherbereich geschaffen, der einen einfachen, schnellen und für den Chip schonenden Zugriff auf die globalen Variablen ermöglicht.

Die Erfindung ist insbesondere für Anwendungsfälle für globale Variablen gut geeignet, bei denen die Variablen zwar global sein müssen, z.B. weil sie über Kontextgrenzen hinaus verfügbar sein müssen, bei denen aber eine lange Lebensdauer der Variablen nicht erforderlich ist. Denn der (schnelle) RAM-Speicher, in dem der Speicherbereich für die Variablen reserviert ist, und in dem die Variablen bei Bedarf angelegt werden, ist flüchtig. Folglich werden die angelegten Variablen gelöscht, sobald die Energieversorgung des RAM-Speichers unterbrochen wird.

Vorzugsweise ist der Variablen-Speicherbereich statisch reserviert, beispielsweise dadurch, dass der Variablen-Speicherbereich für Klassenvariablen reserviert ist (gleichbedeutend: Variablen vom Typ `static`), da der Zugriff auf die Variablen bei einer statischen Reservierung besonders schnell ist. Ein Zugriff (Lesezugriff bzw. Schreibzugriff) auf den statisch reservierten Variablen-Speicherbereich bedeutet, dass eine statisch reservierte Variable (Variable vom Typ static) verwendet (gelesen oder überschrieben) wird. Da bei der Verwendung von statisch reservierten Variablen keine Firewall-Prüfung durchgeführt wird, hat die statische Reservierung des Variablen-Speicherbereichs den zusätzlichen Vorteil, dass der Zugriff auf die Variablen

schneller ist als wenn der Variablen-Speicherbereich dynamisch, während der Laufzeit des Programms reserviert würde. Folglich ist es besonders bevorzugt, dass der Variablen-Speicherbereich statisch reserviert ist.

Weiter ist es bevorzugt, dass solche Programme auf den Variablen-Speicherbereich zugreifen können, die auf den Variablen-Speicherbereich linken können. Dabei ist weiter vorzugsweise das Vermögen eines Programms, auf den Variablen-Speicherbereich linken zu können, dadurch erzielt, dass dem Programm zum Linken eine Export-Komponente auf dem Smart Card Chip zur Verfügung steht. Der Variablen-Speicherbereich erscheint für ein Programm, das darauf zugreifen will, wie ein fremdes Programm. Daher benötigt das Programm, das auf den Variablen-Speicherbereich zugreifen will, die Export-Information des fremden Programms, das den Variablen-Speicherbereich geschaffen hat. Vorzugsweise hat also ein Programm, das den reservierten Variablen-Speicherbereich nutzen möchte, zum Linken die Export-Komponente eines fremden Programms zur Verfügung, das den Variablen-Speicherbereich geschaffen hat.

Weiter sind vorzugsweise solche Programme davon ausgeschlossen sind, den Variablen-Speicherbereich zu nutzen, die nicht auf den Variablen-Speicherbereich linken können. Das Unvermögen eines Programms, auf den Variablen-Speicherbereich linken zu können, ist vorzugsweise dadurch erzielt, dass dem Programm zum Linken eine Export-Komponente des Smart Card Chips vorenthalten ist.

Weiter bevorzugt ist der Variablen-Speicherbereich dadurch reserviert, dass in einem Systemspeicher des Chips ein entsprechendes Java Paket abgelegt

ist, das vorzugsweise nur die Reservierung des Variablen-Speicherbereichs enthält. Ein solches Java Paket mit dem Namen z.B.

„com.gieseckedevrient.javacard.os.commonram",

das Speicherplatz für eine Mehrzahl von statischen Variablen „myRamVarA", „myRamVarB", etc. vom Datentyp `short` und `Byte` und `Int` reserviert, kann beispielsweise folgende Gestalt haben:

```
package com.gieseckedevrient.javacard.os.commonram;
        static short myRamVarA;
        static short myRamVarB;
                etc.
```

Das Java Paket wird im Systemspeicher (ROM, ggf. auch Flash-Speicher) abgespeichert. Bei der späteren Verwendung des Java Pakets werden die entsprechenden Daten, d.h. die statischen, globalen Variablen vom Typ `short`, `Byte`, `Int` im RAM abgelegt.

Weiter ist es bevorzugt, dass auf den Variablen-Speicherbereich nur im Systemspeicher (ROM, Flash) abgespeicherte Programme zugreifen können. Alternativ oder zusätzlich können vorzugsweise auf den Variablen-Speicherbereich nur Programme zugreifen, die bis zum Abschluss der Komplettierung des Smart Card Chips in dem Smart Card Chip implementiert worden sind.

Gemäß einer bevorzugten Ausführungsform wird nur für Pakete, die bis zur Komplettierung implementiert werden, zum Linken die Export-Komponente zur Verfügung gestellt. Ein geladenes Paket, das bis zur Komplettierung

implementiert wird, wird vorzugsweise mit dem Offcard-Linker und der Export-Datei gelinkt. Dadurch wird die Linkinformation zum Linken auf den Variablen-Speicherbereich aus der Export-Komponente des Java Pakets, das den Variablen-Speicherbereich reserviert, in die Import-Komponente des geladenen Pakets geladen. Die Folge ist, dass das geladene Paket auf den reservierten Variablen-Speicherbereich zugreifen kann und dort, im RAM, die RAM-Variablen benutzen kann. Pakete, die nach der Komplettierung in die Java Card bzw. den Chip implementiert werden, haben in ihren Import-Komponenten nicht die Linkinformation aus der Export-Komponente des Java Pakets, das den Variablen-Speicherbereich reserviert hat. Daher können gemäß der bevorzugten Ausführungsform nach der Komplettierung implementierte Pakete die erfindungsgemäßen RAM-Variablen nicht benutzen, d.h. nicht auf den reservierten Variablen-Speicherbereich im RAM zugreifen.

Die virtuelle Java-Card-Maschine ist vorzugsweise derart gestaltet, und dabei bei Bedarf gegenüber der virtuellen Java-Card-Maschine gemäß der Java Card VM Spezifikation modifiziert, dass sie auf Grundlage des im flüchtigen Arbeitsspeicher (RAM) reservierten Variablen-Speicherbereichs die Verwendung von globalen Variablen im RAM ermöglicht.

Insbesondere sind bei Bedarf die Befehle „getstatic" und „putstatic" der virtuellen Java-Card-Maschine gegenüber den Befehlen „getstatic" und „putstatic" virtuellen Java-Card-Maschine gemäß der Java Card VM Spezifikation modifiziert.

Vorzugsweise sind die bei Bedarf vorgenommenen Modifizierungen oder Änderungen an der virtuellen Java-Card-Maschine derart vorgenommen,

dass die Modifizierungen / Änderungen bei der Verwendung der VM nach außen nicht qualitativ erkennbar sind. Dass die Modifizierungen qualitativ nicht erkennbar sind, bedeutet insbesondere, dass der Smart Card Chip einer vorbestimmten Java Card Spezifikation genügt, der der Chip ohne die Modifikationen ebenfalls genügen würde. Quantitativ können die Modifizierungen optional nach außen erkennbar sein, z.B. kann nach außen erkennbar sein, dass das Verwenden der erfindungsgemäßen globalen RAM-Variablen schneller ist als das Verwenden von bekannten globalen Variablen aus dem Stand der Technik. Insbesondere sind die bei Bedarf an den Befehlen „getstatic" und „putstatic" vorgenommenen Änderungen vorzugsweise derart durchgeführt, dass das Verwenden (z.B. Anlegen, Variablenwert ändern) von static Variablen unter Verwendung der Befehle „getstatic" und „putstatic" unverändert ist gegenüber dem Verwenden (z.B. Anlegen, Variablenwert ändern) von static Variablen gemäß der Java Card VM Spezifikation.

Dass die Änderungen nach außen nicht qualitativ erkennbar sind, wird beispielsweise dadurch erreicht, dass ein Variablen-Speicherbereich für globale Variablen im RAM reserviert wird, wobei für die Reservierung eine ansonsten der Java Card VM Spezifikation genügende Variablen-Deklaration verwendet wird, beispielsweise eine Deklaration von static Variablen, nur mit dem Unterschied, dass die static area im RAM statt im EEPROM vorgesehen ist.

Die Einhaltung der Spezifikationen wird vorzugsweise ferner dahingehend gewährleistet, dass ein normaler Benutzer den reservierten Variablen-Speicherbereich nicht benutzen kann, d.h. keine erfindungsgemäßen RAM-

Variablen benutzen kann, da ihm die Linkinformation fehlt, weil ihm die zum Linken erforderliche Export-Datei vorenthalten ist. Beispielsweise wird die Export-Datei beim Hersteller der Smart Card verwahrt und gegenüber Benutzern wie Abnehmern von Smart Cards geheim gehalten. Vorzugsweise können nur Programmentwickler von Systemprogrammen (und wahlweise zusätzlich von preloaded packages), die vor der Komplettierung gelinkt werden, die erfindungsgemäßen RAM-Variablen verwenden, da nur diesen Programmentwicklern von Systemprogrammen (und ggf. preloaded packages) die geheime Export-Datei und somit die zum Linken auf den reservierten Variablen-Speicherbereich erforderliche Linkinformation zur Verfügung steht. Nur Programmentwickler von Systemprogrammen (und ggf. preloaded packages) sind also von der allgemeinen Geheimhaltung der Export-Datei ausgenommen. Abnehmer von Smart Cards, die nach der Komplettierung der Smart Card Programme (z.B. Anwendungen) in die Smart Card (bzw. in den Smart Card Chip) laden, können zum Linken hingegen nur den Oncard-Linker verwenden, der die Linkinformation zum Linken auf den reservierten Variablen-Speicherbereich im RAM nicht hat.

Im folgenden wird die Erfindung anhand von Ausführungsbeispielen und unter Bezugnahme auf die einzige Figur 1 der Zeichnung näher erläutert.

Fig. 1 zeigt eine schematische Darstellung einer Java Card mit einem darin implementierten RAM package 5 , durch das ein Variablen-Speicherbereich 18 im RAM der Java Card statisch reserviert ist, gemäß einer bevorzugten Ausführungsform der Erfindung.

Die Java Card aus Fig. 1 hat einen nichtflüchtigen Systemspeicher 1, nämlich den ROM 1, einen nichtflüchtigen Anwendungsspeicher 2, nämlich den EEPROM 2, und einen flüchtigen Arbeitsspeicher 3, nämlich den RAM 3.

Im ROM 1 sind die virtuelle Java-Card-Maschine VM 4 und weiterer nativer Code implementiert. Zudem sind im ROM 1 eine Reihe von Paketen Pckg1, Pckg2, ... Pckgn mit unterschiedlichen Inhalten, zum Teil Systemfunktionen und zum Teil Anwendungen, implementiert. Außerdem ist im ROM 1 das erfindungsgemäße RAM package 5 implementiert, also das Java Paket, durch welches der Variablen-Speicherbereich 18 im RAM reserviert ist. Im EEPROM 2 sind mehrere vor der Komplettierung in die Java Card geladene Pakete mit Anwendungen, nämlich mehrere preloaded EEPROM packages 16, abgespeichert. Zudem enthält der EEPROM 2 einen heap-Speicher 19 mit dynamischen Daten (Dynamic Data), mehrere Pufferspeicher (Buffers) und einen Bereich mit nativen Daten. Der RAM 3 enthält native RAM-Daten, den durch das RAM package reservierten Variablen-Speicherbereich 18 (RamP.-Data), zwei Pufferspeicher (Buffer), einen Arbeitsspeicherbereich, der auf Anfrage gelöscht wird (COD, Clear On Demand) und einen Arbeitsspeicherbereich, der bei Reset, also bei Rücksetzen, gelöscht wird (COR, Clear On Reset).

Die virtuelle Maschine VM 4 in Fig. 1 enthält beispielsweise die Befehle der VM 4 wie z.B. „putstatic" 7, mit dem sich eine static Variable zur Laufzeit ändern lässt, wie durch den vergrößerten Ausschnitt 7 der VM mit der Beschriftung „Putstatic" angedeutet ist.

Im beschriebenen Beispiel von Fig. 1 gibt es weiter eine in der Java Card implementierte Zugriffstabelle (SAT, segment allocation table), die u.a. die Anfangsadressen der in der Java Card implementierten Pakete (packages) enthält.

Die Zugriffstabelle SAT enthält insbesondere für das RAM package einen static area descriptor 8, d.h. einen in dem RAM package 5 enthaltenen Deskriptor 8 (oder eine Angabe), der einen Anfang und eine Größe eines bestimmten Adressbereichs als Speicherbereich für statisch deklarierte (reservierte) Variablen angibt. Der Deskriptor 8 (static area descriptor) enthält zwei Teilangaben, nämlich die Anfangsadresse des Adressbereichs und die Größe des Adressbereichs. Die Anfangsadresse ist so gewählt, dass der durch den static area descriptor 8 bestimmte Adressbereich im RAM 3 der Java Card liegt, wie durch die Pfeile 12, 13 angedeutet ist, die vom Deskriptor 8 (static area descriptor) zum RAM 3 verlaufen. Durch die derart getroffene Auswahl der Anfangsadresse (d.h. Auswahl der Anfangsadresse so, dass der reservierte Variablen-Speicherbereich im RAM liegt) ist eine direkt auf den RAM-Speicher gerichtete Zugriffsinformation gebildet. Dass der Deskriptor 8 (static area descriptor) auf eine Adressangabe im RAM verweist, stellt eine Änderung gegenüber der Java Card VM Spezifikation dar, die jedoch nach außen nicht erkennbar ist.

Das in Fig. 1 dargestellte Paket 6 mit der Nummer „n", bezeichnet mit „Pckgn", ist ein preloaded package, d.h. ein Paket, das vor der Komplettierung der Java Card in der Java Card implementiert worden ist, und enthält im ROM-Speicher 1 der Java Card abgelegten Paket-Code 9, wie durch den vergrößerten Ausschnitt 9 „Preloaded Pack. Code" angedeutet ist. Der Pa-

ket-Code 9 des preloaded package 6 wiederum enthält Code für einen Konstantenpool 10 („Const. Pool") und Code 11 für den Befehl „putstatic" der VM. Durch „putstatic" 11 im Code des preloaded package 6 (Pckgn) wird unter Verwendung des Konstantenpools 10 eine Variable vom Typ static adressiert, und zwar im RAM 3, im reservierten Variablen-Speicherbereich 18, wie durch die Pfeile 14, 15 angedeutet ist, die von „Code putstatic" zu „Const. Pool ..." (Pfeil 14) und von „Const. Pool ..." zu „RamP. Data" (Pfeil 15) verlaufen. Das preloaded package Paket Pckgn 6 wurde durch den Off-card-Linker und unter Verwendung der Export-Datei der Java Card gelinkt. Dabei wurde das preloaded package Paket Pckgn 6 mit Linkinformation des RAM package 5 ausgestattet, insbesondere mit der Adresse, die im Deskriptor 8 für static Variablen (static area descriptor) steht. Hierdurch hat das preloaded package Pckgn 6 die Linkinformation des RAM package 5. Folglich kann das preloaded package Pckgn 6 den reservierten Variablen-Speicherbereich 18 des RAM package 5 benutzen, also erfindungsgemäße globalen Variablen (RAM static Variablen) benutzen.

Bei dem Beispiel aus Fig. 1 kann ein postloaded package 16, das im EEPROM implementiert ist, nicht auf den im RAM 3 reservierten Variablen-Speicherbereich 18 zugreifen, so dass das postloaded package 16 im EEPROM keine erfindungsgemäßen globalen Variablen im RAM benutzen kann. Nur Pakete im Systemspeicher ROM können die globalen Variablen im RAM benutzen.

Gemäß alternativen Ausführungsformen können auch preloaded packages im EEPROM die globalen Variablen im RAM benutzen. Die preloaded packages im ROM haben dann, wie die packages im Systemspeicher ROM

auch, die vom Offcard-Linker in der Export-Datei bereitgestellte Adressinformation, mit der sie auf den reservierten Variablen-Speicherbereich linken können, so dass sie den reservierten Variablen-Speicherbereich im RAM für Variablen benutzen können.

Patentansprüche

1. Smart Card Chip mit einem nichtflüchtigen Systemspeicher (ROM, Flash1), einer in dem nichtflüchtigen Systemspeicher (ROM, Flash1) implementierten virtuellen Java-Card-Maschine, einem nichtflüchtigen Anwendungsspeicher (EEPROM, Flash2), einem flüchtigen Arbeitsspeicher (RAM) und einem für globale Variablen reservierten Variablen-Speicherbereich, wobei der Variablen-Speicherbereich im flüchtigen Arbeitsspeicher (RAM) reserviert ist.

2. Smart Card Chip nach Anspruch 1, wobei auf den Variablen-Speicherbereich nur im Systemspeicher (ROM, Flash) abgespeicherte Programme zugreifen können.

3. Smart Card Chip nach Anspruch 1 oder 2, wobei auf den Variablen-Speicherbereich nur Programme zugreifen können, die bis zum Abschluss der Komplettierung des Smart Card Chips in dem Smart Card Chip implementiert worden sind.

4. Smart Card Chip nach einem der Ansprüche 1 bis 3, wobei der Variablen-Speicherbereich statisch reserviert ist.

5. Smart Card Chip nach einem der Ansprüche 1 bis 4, wobei der Variablen-Speicherbereich durch eine direkt auf den Arbeitsspeicher (RAM) gerichtete Zugriffsinformation reserviert ist.

6. Smart Card nach einem der Ansprüche 1 bis 5, wobei solche Programme auf den Variablen-Speicherbereich zugreifen können, die auf den Variablen-Speicherbereich linken können.

7. Smart Card Chip nach Anspruch 6, wobei das Vermögen eines Programms, auf den Variablen-Speicherbereich linken zu können, dadurch erzielt ist, dass dem Programm zum Linken eine Export-Komponente auf dem Smart Card Chip zur Verfügung steht.

8. Smart Card Chip nach einem der Ansprüche 1 bis 7, wobei solche Programme davon ausgeschlossen sind, den Variablen-Speicherbereich zu nutzen, die nicht auf den Variablen-Speicherbereich linken können.

9. Smart Card Chip nach Anspruch 8, wobei das Unvermögen eines Programms, auf den Variablen-Speicherbereich linken zu können, dadurch erzielt ist, dass dem Programm zum Linken eine Export-Komponente des Smart Card Chips vorenthalten ist.

10. Smart Card Chip nach einem der Ansprüche 1 bis 9, wobei der Variablen-Speicherbereich durch ein in dem Smart Card Chip implementiertes Java Paket (RAM package) reserviert ist.

11. Smart Card Chip nach Anspruch 10, wobei das Java Paket (RAM package) im Systemspeicher (ROM, Flash1) implementiert ist.

12. Smart Card Chip nach Anspruch 10 oder 11, wobei das Java Paket ausschließlich die Reservierung des Variablen-Speicherbereichs enthält.

13. Smart Card Chip nach einem der Ansprüche 10 bis 12, wobei eine Export-Komponente des Java Pakets (RAM package), in der die zum Linken

auf den reservierten Variablen-Speicherbereich erforderliche Linkinformation enthalten ist, nicht in dem Smart Card Chip implementiert ist.

14. Smart Card Chip nach einem der Ansprüche 1 bis 13, wobei die virtuelle Java-Card-Maschine derart gestaltet ist, und dabei bei Bedarf modifiziert ist, dass sie auf Grundlage des im flüchtigen Arbeitsspeicher (RAM) reservierten Variablen-Speicherbereichs die Verwendung von globalen Variablen im RAM ermöglicht.

15. Smart Card Chip nach Anspruch 14, wobei die virtuelle Java-Card-Maschine derart modifiziert ist, dass die Modifizierungen nach außen hin nicht qualitativ erkennbar sind, insbesondere dass der Smart Card Chip einer vorbestimmten Java Card Spezifikation genügt, welcher der Chip ohne die Modifikationen ebenfalls genügen würde.

16. Chipmodul mit einem Smart Card Chip gemäß einem der Ansprüche 1 bis 15.

17. Datenträger, insbesondere Java Card, mit einem Smart Card Chip gemäß einem der Ansprüche 1 bis 15 und/oder einem Chipmodul gemäß Anspruch 16.

18. Verfahren zum Reservieren eines Variablen-Speicherbereichs in einem Smart Card Chip, der aufweist: eine nichtflüchtigen Systemspeicher (ROM, Flash1), eine in dem nichtflüchtigen Systemspeicher (ROM, Flash1) implementierte virtuelle Java-Card-Maschine, einen nichtflüchtigen Anwendungsspeicher (EEPROM, Flash2) und einen flüchtigen Arbeitsspeicher (RAM),

wobei bei dem Verfahren ein Java Programmcode in dem Smart Card Chip implementiert wird, durch den ein Variablen-Speicherbereich für globale Variablen im flüchtigen Arbeitsspeicher (RAM) reserviert wird.

19. Verfahren nach Anspruch 18, wobei als Java Programmcode eine Java Paket (RAM package) implementiert wird.

20. Verfahren nach Anspruch 19, wobei eine Export-Komponente des Java Pakets, in der die zum Linken auf den Variablen-Speicherbereich erforderliche Linkinformation enthalten ist, nicht in dem Smart Card Chip implementiert wird.

21. Verfahren nach einem der Ansprüche 18 bis 20, wobei eine zum Linken auf den reservierten Variablen-Speicherbereich erforderliche Export-Datei nur solchen Programmen zum Linken zur Verfügung gestellt wird, die auf den Variablen-Speicherbereich zugreifen können sollen.

Zusammenfassung

Die Erfindung schafft einen Smart Card Chip mit einem nichtflüchtigen Systemspeicher (ROM, Flash1), einer in dem nichtflüchtigen Systemspeicher (ROM, Flash1) implementierten virtuellen Java-Card-Maschine, einem nichtflüchtigen Anwendungsspeicher (EEPROM, Flash2), einem flüchtigen Arbeitsspeicher (RAM) und einem für globale Variablen reservierten Variablen-Speicherbereich, wobei der Variablen-Speicherbereich im flüchtigen Arbeitsspeicher (RAM) reserviert ist. Der Variablen-Speicherbereich ist vorzugsweise statisch reserviert. Die Verwendung der Variablen kann auf Systempackages und wahlweise zudem auf preloaded (ROM/ EEPROM) packages beschränkt sein.

Figur 1

RAM package

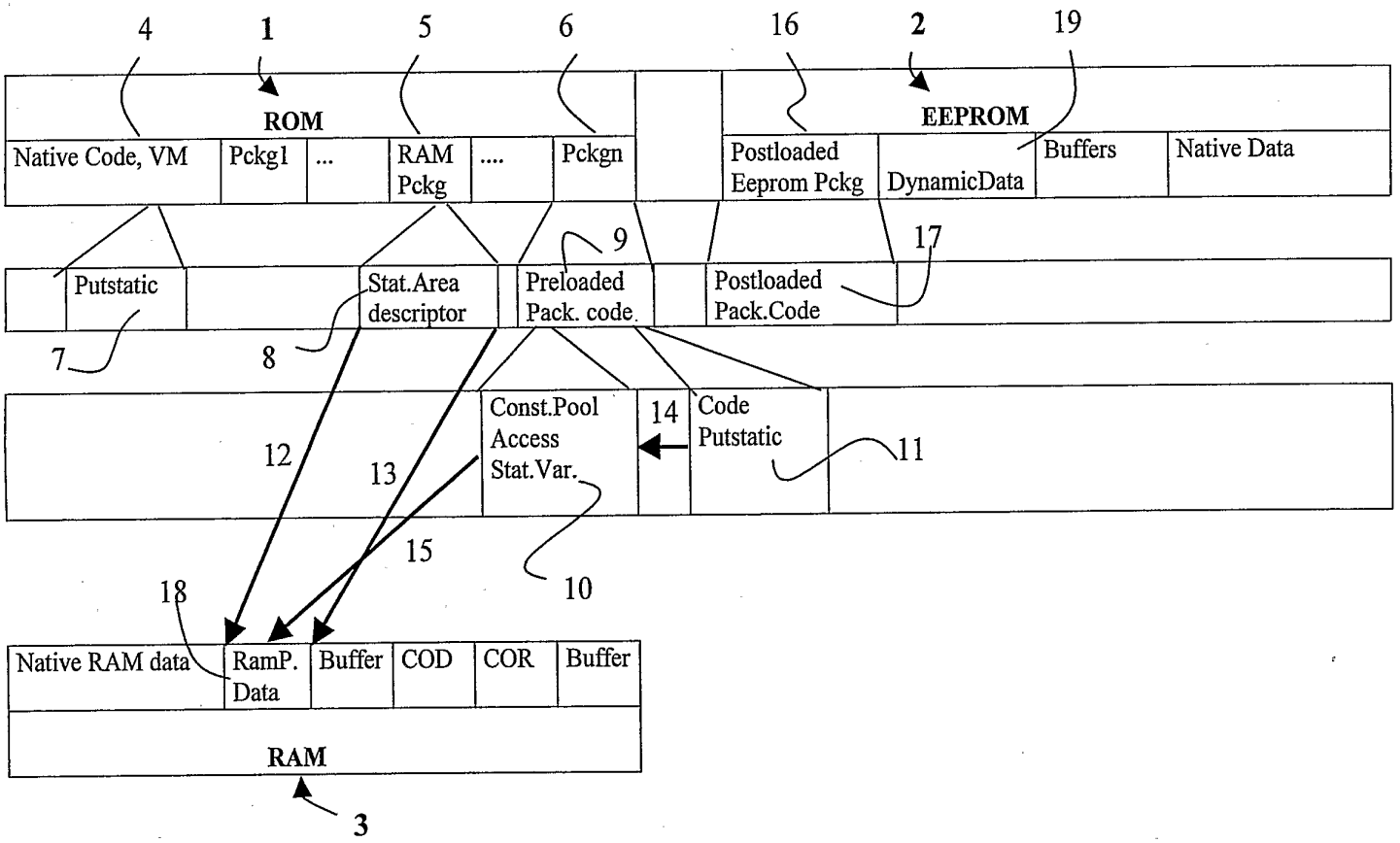


Fig. 1